

# DNSim Overview

## What is DNSim?

The Dynamic Neural Simulator (DNSim) is a tool that helps users build and explore large differential equation models; it is coupled to a database of existing neural models that can serve as a starting point for building new models. It can be used in Matlab as a toolbox and graphical interface or as a standalone application using Matlab's free MCR. The power of DNSim comes from its ability to manipulate models that can be structured in a particular way especially useful for modeling real-world systems. The generic structure supports modular modeling and reuse of existing model components; DNSim makes this practical and easy to do. It also provides other features that simplify the modeling and simulation process.

## Feature overview

- **Dynamic simulation:** *real-time simulation that continues running as you interactively (dynamically) adjust parameters*
- **Model remixing:** *combine select components from two or more models*
- **Model comparison:** *list all differences between two models*
- **Model-tagged Note taking:** *record notes during model building or simulation with the subsequent ability to revert back to any earlier noted model*
- **Interface to InfiniteBrain model database:** *download existing models from/upload new models to DB*
- **Cluster computing and batch management:** *manage simulation batches that vary parameters or model components; automate cluster job creation and submission using qsub from a login node.*

Next: Get started with the Quick Start Guide.

# DNSim Quick Start Guide

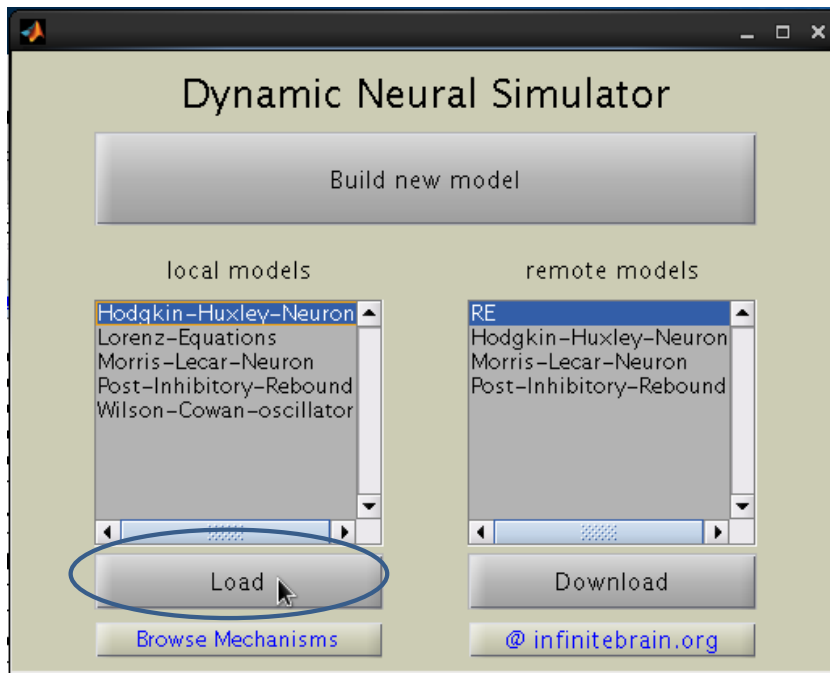
## 1. Install and run DNSim

Using Linux with Matlab and git: (enter at terminal)

```
git clone https://github.com/jsherfey/dnsim.git
cd dnsim
matlab -r dnsim
```

Other: see <http://infinitebrain.org/setup/>

## 2. Select and load (download) a model



- Local models include all nodes/networks stored in the dnsim/database directory.

- Click “Browse Mechanisms” to view existing mechanism models.

- Right-click on remote model name to open model page in default web browser.

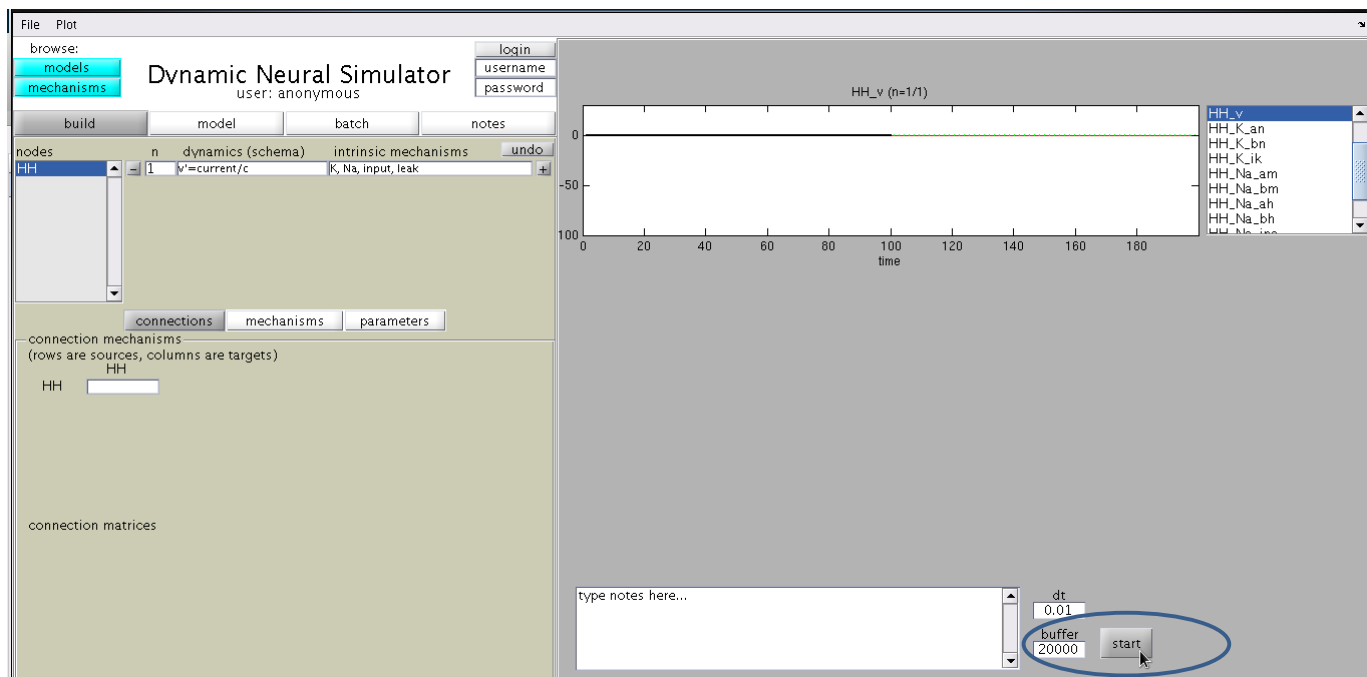
- Remote InfiniteBrain repository:

<http://infinitebrain.org/models/>

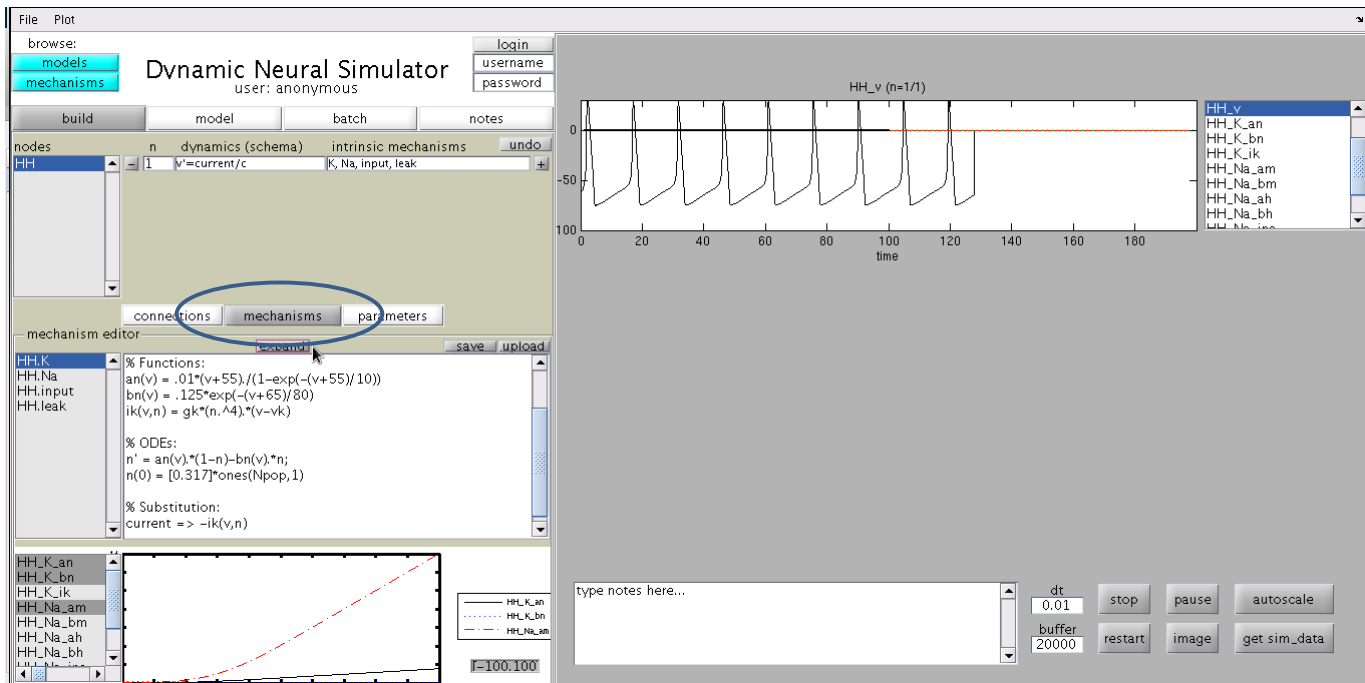
- If you get “FTP Error: 425” when downloading/uploading models, enter in Matlab Command Window:

`rehash toolboxcache`

## 3. Start simulating the model



## 4. Explore and modify the model



- Model equations and parameters can be interactively tuned during ongoing simulation.
- Auxiliary functions can be viewed and interactively tuned in the mechanism tab.
- For cluster computing: add dnsim/csh to shell environment path (e.g., “export PATH=\$PATH:~/dnsim/csh”).

### Further recommendations:

- Next: Load/download and explore a few models to get familiar with the interface. Then read the User Manual to learn how to implement your models (see process overview below).

Create an InfiniteBrain account. You can log in to the web site and DNSim tool with that account to save and share your own models through the InfiniteBrain. Sign up at <http://infinitebrain.org/>.

## Interface

### Main tabs:

- build tab: build/modify your model
  - connections tab: list mechanisms connecting source nodes (rows) to target nodes (columns)
  - mechanisms tab: access/edit mechanism equations
  - parameters tab: set node model parameters. Recommendation: set mechanism parameters in mechanisms tab and parameters of the node dynamics (schema) in the parameters tab.
- model tab: view a printout of the model components and equations
- batch tab: run sets of simulations varying parameters (locally or on a cluster)
- notes tab: review notes taken in the note box (next to simulation controls) and compare noted models

## Build process

Start: from Launch screen: click “Build new model”.

1. Conventional approach: directly enter full model equations (contain in a mechanism)
    1. Under Build tab: Input new mechanism name in the intrinsic mechanism list
    2. Under mechanisms tab: expand the mechanism editor and enter your model equations
  2. Modular approach: build from existing sub-models (using nodes and mechanisms)
    1. Under Build tab:
      - a. set the node dynamics (schema)
      - b. Input intrinsic and connection mechanisms (tip: browse & use existing mechanisms)
    2. Under mechanisms tab: input/edit mechanism equations and parameters
- Save model to disk (File → Save model) or upload to InfiniteBrain (File → Upload model)

Infinite Brain
jasonsherfey
Sign in

# The Infinite Brain

Uniting modelers and experimentalists across neuroscience.

Sign up »

## DNSim Setup

A modular modeling tool.

View on GitHub »

## Enter the DB

Database of existing models.

## News

- Infinite brain news  
August 27, 2014
- first tweet test  
August 13, 2014

© 2014 - Boston University. Site and software produced by Jason Sherfey.

Repository: <http://infinitebrain.org/models/>

Welcome, jasonsherfey! Sign out

## Models [evolution]

rating	name	level	author	date-added	composition
1	Hodgkin-Huxley-Neuron	node	anonymous	Sept. 18, 2014, 3:58 p.m.	
0	ML_Ca	mechanism	anonymous	Sept. 18, 2014, 4:34 p.m.	
0	HH_Na	mechanism	anonymous	Sept. 18, 2014, 4:33 p.m.	
0	HH_K	mechanism	anonymous	Sept. 18, 2014, 4:32 p.m.	
0	RE	network	anonymous	Sept. 18, 2014, 4:01 p.m.	
0	Post-Inhibitory-Rebound	node	anonymous	Sept. 18, 2014, 4 p.m.	
0	Morris-Lecar-Neuron	node	anonymous	Sept. 18, 2014, 4 p.m.	

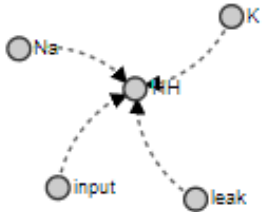
Model detail page:

Infinite Brain

Welcome

# Hodgkin-Huxley-Neuron

- node model uploaded by anonymous at Sept. 18, 2014, 3:58 p.m.
- [node equations](#)
- Tags: squidaxon, HH,
- Notes: Hodgkin-Huxley model
- [dsim specification](#)



## Discuss

Add comment

- D3 graph shows model composition (dotted lines link mechanisms to nodes; solid lines link nodes to nodes)
- Comments can be used to discuss your model with others who have InfiniteBrain accounts
- Tags and Notes can be set when uploading your model in DNsims or from InfiniteBrain.org user dashboard

# DNSIM User Manual

## What is DNSim?

DNSim (the Dynamic Neural Simulator) is a tool for rapid prototyping of large first-order systems of ODEs. It is coupled to a database of neural models that can be downloaded, combined, and used as a starting point for building new models. The database can be accessed at [InfiniteBrain.org](http://InfiniteBrain.org) or within the tool's graphical interface.

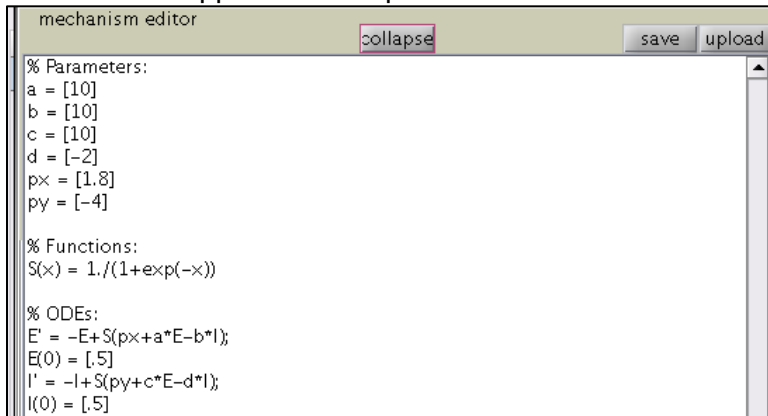
## Building models

Models can be specified in two ways:

- The conventional approach: listing all parameters and equations in one text box
- The modular approach (described below)

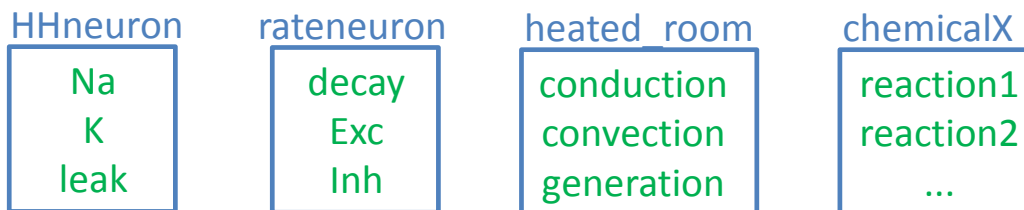
The conventional approach is straightforward and best suited for simple systems of differential equations.

Conventional approach example: Wilson-Cowan model of interacting excitatory and inhibitory populations:



- Comments begin with “%”
- Built-in matlab functions are recognized

The strength of DNSim for building large models comes with the modular approach. It is best suited for modeling physical systems that can be described in terms of interacting parts (nodes) with dynamics determined by some mechanisms. Nodes could be neurons, compartments, chemicals, volumes, or any other system with a state and dynamics describing how the state changes over time. Mechanisms are things that affect how that state changes over time. Example nodes and the mechanisms that affect them:

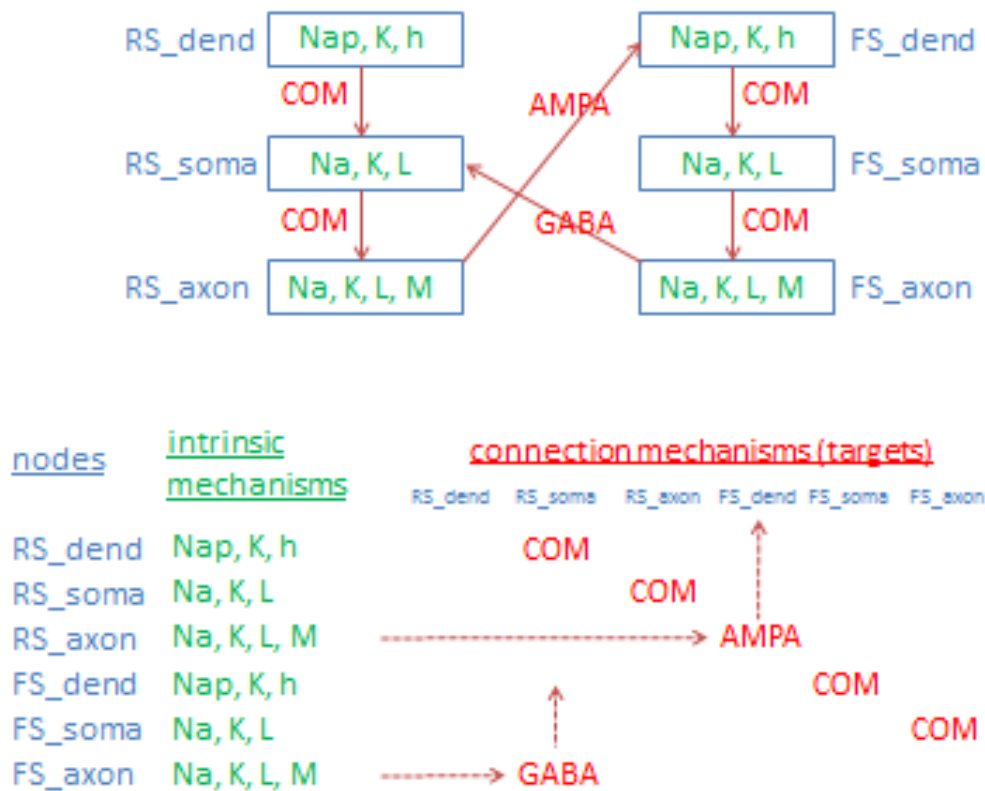


In the modular approach, DNSim processes a higher-level specification of the model structure given schemas for the node dynamics and mechanisms defining terms that plug into those schemas. The result is the automatic construction of the full system of equations from a less complex specification of the model.

Mechanism (sub)models are modular in the sense that they can be defined once and then be reused in multiple nodes; any number of mechanisms may contribute to the dynamics of a given node. For instance, two neural compartments may have sodium (Na) currents with the same kinetics; in this case, it would be best to define the sodium current model once, add it to both compartments, and then set the mechanism parameters separately for each compartment. Whether stored on disk or in the InfiniteBrain repository, mechanisms and nodes that have been previously created can be easily incorporated into new models.

Next, the modular approach and its benefits will be illustrated with an example. Then, the system generation process will be described, and a general procedure for building models with the modular approach will be given.

Modular approach example: interacting populations of multicompartment excitatory and inhibitory neurons.



The schematic representation of the model structure (upper diagram) can be reorganized into the lower tabular form which maps clearly onto DNSim controls in the following way:

build	model	batch	notes
nodes	n	dynamics (schema)	intrinsic mechanisms
RS_dend	5	$V' = \text{current} / C_m$	Na <sub>p</sub> , K, h, input
RS_soma	5	$V' = \text{current} / C_m$	Na, K, leak
RS_axon	5	$V' = \text{current} / C_m$	Na, K, M, leak
FS_dend	1	$V' = \text{current} / C_m$	Na <sub>p</sub> , K, h
FS_soma	1	$V' = \text{current} / C_m$	Na, K, leak
FS_axon	1	$V' = \text{current} / C_m$	Na, K, M, leak

connections

mechanisms

parameters

connection mechanisms  
(rows are sources, columns are targets)

	RS_dend	RS_soma	RS_axon	FS_dend	FS_soma	FS_axon
RS_dend		COM				
RS_soma			COM			
RS_axon				AMPA		
FS_dend					COM	
FS_soma						COM
FS_axon		GABA <sub>A</sub>				

This is a sufficient specification of the full model if the mechanism sub-models already exist. Search for existing mechanisms in the online InfiniteBrain repository or in the DNSim Mechanism Browser:

Browse Mechanisms					
	name	site	notes	local	id
1	HH_K	link	Hodgkin...	<input type="checkbox"/>	56
2	HH_Na	link	Hodgkin...	<input type="checkbox"/>	57
3	ML_Ca	link	Morris-L...	<input type="checkbox"/>	58
4	CaDyn			<input checked="" type="checkbox"/>	0
5	CaHVA			<input checked="" type="checkbox"/>	0
6	dgKS			<input checked="" type="checkbox"/>	0
7	dgKdr			<input checked="" type="checkbox"/>	0
8	dgNaf			<input checked="" type="checkbox"/>	0
9	iA			<input checked="" type="checkbox"/>	0
10	iAPoirazi			<input checked="" type="checkbox"/>	0
11	iAR			<input checked="" type="checkbox"/>	0
12	iCOM			<input checked="" type="checkbox"/>	0
13	iCa			<input checked="" type="checkbox"/>	0
14	iCaH			<input checked="" type="checkbox"/>	0
15	iCaTPoirazi			<input checked="" type="checkbox"/>	0
16	iCan			<input checked="" type="checkbox"/>	0
17	iCAP			<input checked="" type="checkbox"/>	0
18	iHVA			<input checked="" type="checkbox"/>	0
19	iK			<input checked="" type="checkbox"/>	0
20	iKDR			<input checked="" type="checkbox"/>	0
21	iKs			<input checked="" type="checkbox"/>	0
22	iL			<input checked="" type="checkbox"/>	0
23	iM			<input checked="" type="checkbox"/>	0
24	iNMDA			<input checked="" type="checkbox"/>	0
25	iNMDAgbar			<input checked="" type="checkbox"/>	0
26	iNa			<input checked="" type="checkbox"/>	0
27	iNaF			<input checked="" type="checkbox"/>	0
28	iNap			<input checked="" type="checkbox"/>	0

```

% Parameters:
gM = [0.75]
E_M = [-95]
c_MaM = [1]
c_MbM = [1]
IC = [0]
IC_noise = [0]

% Functions:
aM(V) = c_MaM.*(0.02./(1+exp((-20-V)/5)))
bM(V) = c_MbM.*(0.01*exp((-43-V)/18))
IM(V,m) = gM.*m.*(V-E_M)

% ODEs:
mM' = (aM(V).*(1-mM)-bM(V).*mM);
mM(0) = IC+IC_noise.*rand(Npop,1)

% Substitution:
current => -IM(V,mM)

```

Any mechanism that does not exist already can be defined by inputting its parameters and equations in the mechanism tab after being listed in either an intrinsic or connection mechanism list (under the build tab). Recommendation: if possible, start with a similar mechanism from the Mechanism Browser. Once a model has been built, mechanisms can be added or removed simply by adding or removing their labels from the appropriate intrinsic and connection mechanism lists (e.g., adding an M-current to a spiking neuron model).

Automated ODE system generation:

DNSim combines node-specific ODE schemas and associated mechanism lists to construct the full system of ordinary differential equations; i.e.,  $X'=F(X)=\text{build}(\text{schema}, \text{mechanisms})$ . An ODE schema is an ODE with placeholder variables to be substituted by terms defined in mechanisms. Specific mechanism terms are linked to placeholder variables in a schema by a substitution statement in the mechanism definition. Example:

Dynamics:  $V' = \text{current}$

Mechanisms: iNa, iK

iNa with current  $\Rightarrow \text{INa}(V, m, h)$

iK with current  $\Rightarrow \text{IK}(V, n)$

Substitutions:

$V' = \text{INa}(V, m, h) + \text{current}$

$V' = \text{INa}(V, m, h) + \text{IK}(V, n) + \text{current}$

Placeholders are removed before simulation.

General procedure for modular modeling in DNSim:

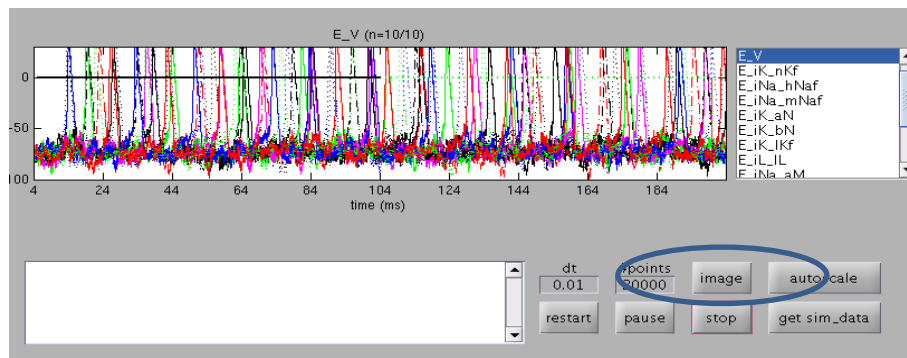
1. Diagram model structure (nodes, networks, mechanisms)
2. Find/create mechanisms and ODE schema
3. Specify nodes and connections between nodes
4. Adjust parameters for each node



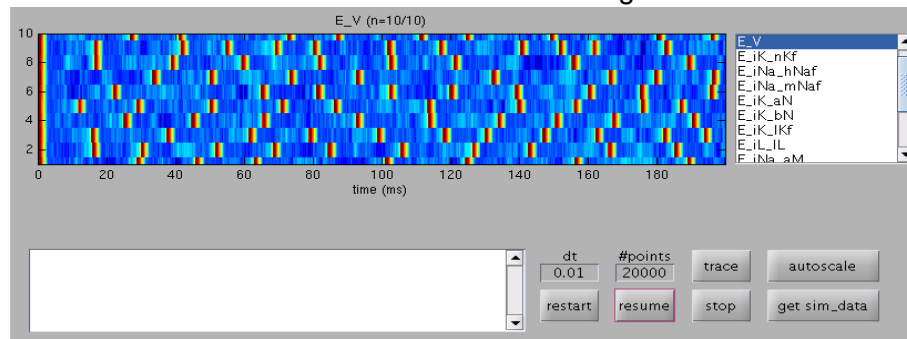
## DNSim features (notes, batch simulation, visualization)

Interactive simulations:

simulation: state variable traces (overlay)



simulation: state variable image



Batch simulations:

batch simulation

machine ☒ local ☐ cluster memlimit 8G

timelimits [0 40]

solver euler dt 0.01

#repeats 1

search space

scope	variable	values
-		

help

outputs

rootdir /usr3/graduate/sherfey/dnsim/database/private dsfact 1

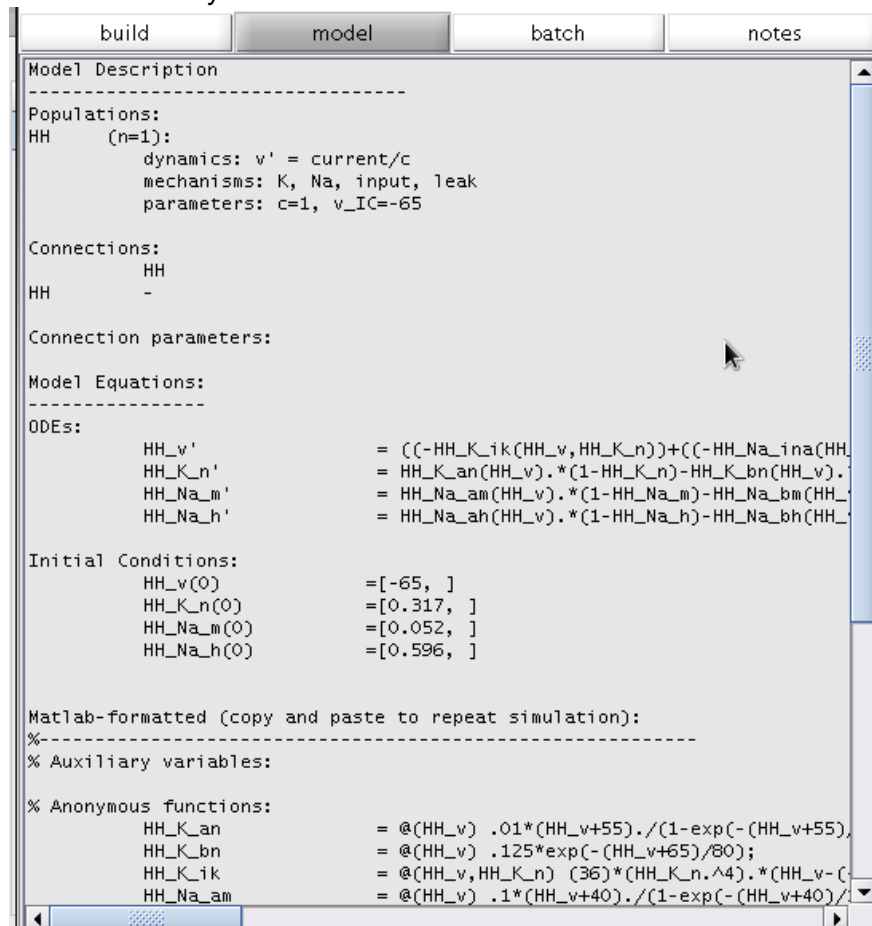
save ☐ data ☐ popavg ☐ spikes ☐ plots

plot ☒ state vars ☐ spike rates ☐ spectrum

submit!

- Output directory structure
  - Cluster management
  - Search space syntax
    - scope = scope of thing to vary (cell or connection label; e.g., 'E', 'I-E', '(E,I)')
    - variable = thing to vary (e.g., 'gNa'; reserved options: 'multiplicity', 'mechanisms')
    - values = values of the thing to vary (e.g., [5:5:50], {'iNap','iK','iM'})
    - special case: variable mechanisms are concatenated/added to those in those in the base model.
- syntax (for arbitrary elements a,b,c):
- [a,b,c] = iterate over set
  - (a,b,c) = group elements
  - {a,b,c} = permute elements
- note: all three set options are optional and available for scopes, variables, and values

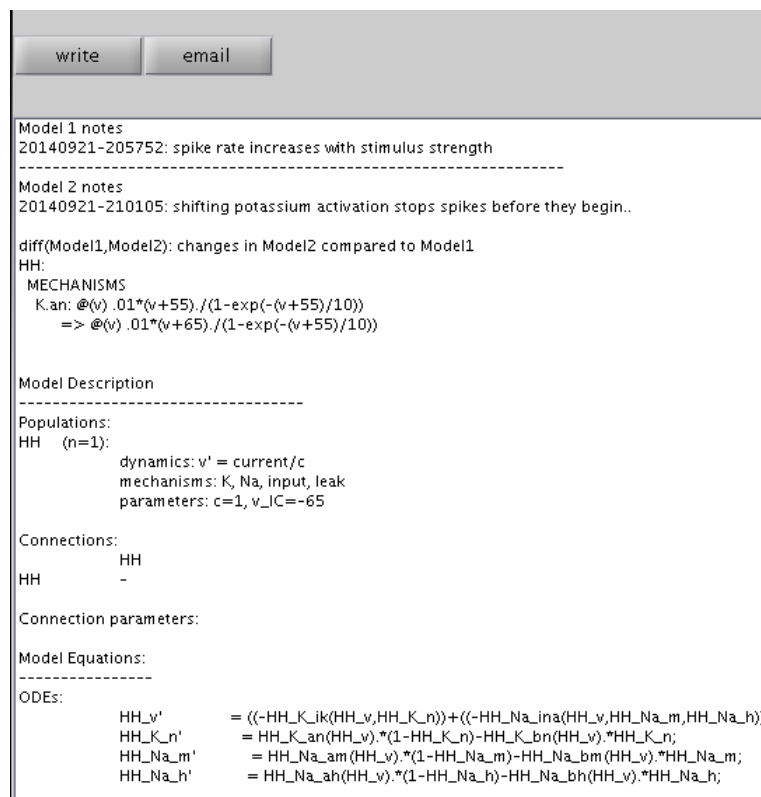
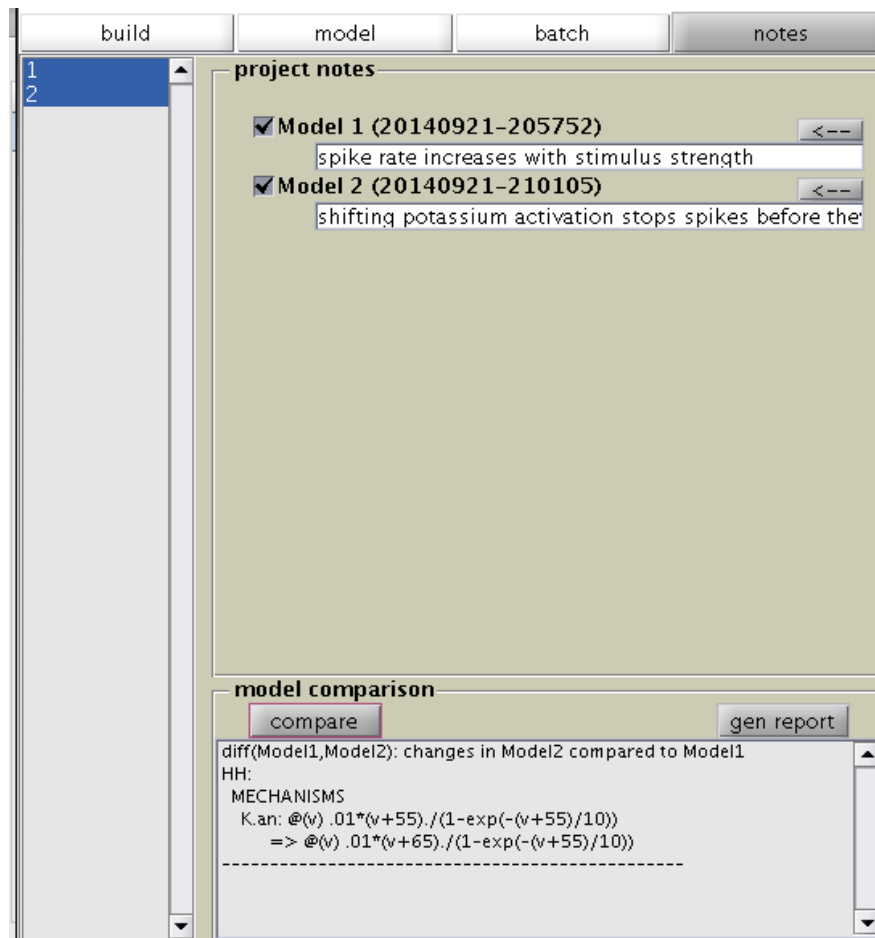
Review model equations for the full system in the `model` tab:



Tip: the model function handle at the end can be copied and used in Matlab for manual simulation.

Note taking:

DNSim provides a edit box in the simulation panel that can be used to take notes at any point. All notes are recorded and can be reviewed under the `notes` tab. The active model at the moment the note is recorded is stored with the note and, at any point, users can revert back to older models by clicking the “←” button on the `notes` tab. The differences in two noted models can be displayed by selecting the checkboxes next to their notes and clicking the “compare” button under the model comparison section of the notes tab. A report listing all notes, the progressive changes between different versions of a model, and the complete model description of the active model can be generated by clicking “gen report”.



The report can be saved to disk or emailed with an attached model file that can be loaded in DNSim.

...

## Multiple ways of inputting models in DNSim

### Simple ODE system

#### Lorenz – dummy mechanism

```

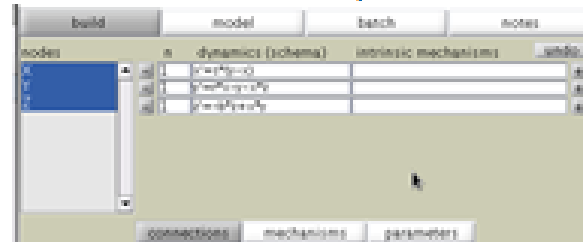
build    model    batch    notes
mechanism editor    collapse    save    upload

% Parameters:
a = [10]
r = [2.7]
b = [2.664]

% ODEs:
x' = r*(y-x)
y' = x*(r-y-z)
z' = -b*(x+y-z)
x(0) = [1]
y(0) = [2]
z(0) = [3]

```

#### Lorenz – node dynamics



### More complicated system

#### Morris-Lecar – dummy mechanism

```

build    model    batch    notes
mechanism editor    collapse    save    upload

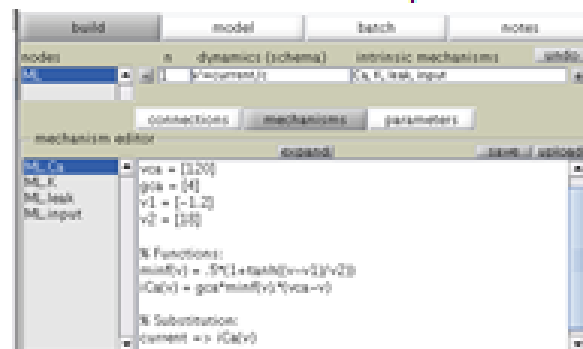
% Parameters:
vk = -84; vl = -60; vca = 120;
i = 100; c = 20;
gk = 8; gl = 2; gca = 4;
v1 = -1.2; v2 = 18; v3 = 2; v4 = 30;

% Functions:
minf(v) = .5*(1+tanh((v-v1)/v2))
winf(v) = .5*(1+tanh((v-v3)/v4))
lame(v) = .04*cosh((v-v3)/(2*v4))

% ODEs:
v' = (i+g*(vl-v)+gk*w*(vk-v)+gca*minf(v)*vca-v)/c;
w' = lame(v)*(winf(v)-w);
w(0) = [0.014873]

```

#### Morris-Lecar – modular specification



## Who should use DNSim?

For computationalists -- If your model is so big that it takes effort to manage and is an ODE system at its core, then this tool can help you. Also, if you want a tool that makes it easy to track your development, share your models, and manage batch simulations locally or on a cluster, this tool might be for you.

For experimentalists and novices -- DNSim enables you to explore and combine components of existing models without a strong background in math or computer programming.

*For neural modelers*, if you are working with a morphologically-realistic, many-compartment spiking neuron, you should start with NEURON or GENESIS. DNSim does not have an explicit representation of space, which makes complex spatial relations difficult to implement. However, if you are working with single or few-compartment rate or spiking neurons or networks of them, especially models including many ionic/other processes, you should consider using this tool. Furthermore, the current version does not check for consistency of physical units; that is up to the user.

## What is DNSim?

DNSim (the Dynamic Neural Simulator) is a tool for rapid prototyping of large first-order systems of ODEs. It is coupled to a database of neural models that can be downloaded, combined, and used as a starting point for building new models. The database can be accessed at InfiniteBrain.org or within the tool's graphical interface.

DNSim is written in Matlab and can be used as a Matlab toolbox (if you have Matlab) or standalone application (if you don't). The toolbox provides access to the underlying functions that enable script-based model building (in addition to model building using the graphical interface). The graphical interface provides a portal to the DNSim database and your InfiniteBrain.org account, if you have one.

In DNSim, models may be built by the standard approach of typing out the full ODE system:  $X' = F(X)$ . However, a much greater benefit can be achieved by utilizing its ability to automatically construct larger ODE systems using a simple specification given smaller models that have already been defined. Smaller models can be downloaded from the DNSim database, loaded from disk, or created from scratch within the graphical interface. This results in a modular approach that makes it easy to build new models from parts of existing models, thus facilitating rapid prototyping of large systems. For those lacking a math background, the simple specification based on existing models also enables large models to be built from meaningful labels without needing to enter equations. This model building approach will be illustrated using a couple examples then outlined to guide you through building your own models.

## Model structure

Let's define some things before discussing how to cast your models in a modular form and how to build models that are represented in that form.

Definitions:

- **Node** = a system or subsystem with a state and dynamics describing how that state changes over time.
  - Examples: point neuron, neuron compartment, chemical species, or brain volume.
  - Notes: nodes have N copies, and thus, may represent single elements (N=1, default) or a population of elements (N>1) with equivalent models (caveat: population heterogeneity can be introduced by assigning parameter values to individual elements using independent draws from probability distributions via built-in Matlab functions; see examples).
- **Connection** = a specification of how one node (the source) affects another node (the target)
- **Network** = the directed graph formed by connections
- **Mechanism** = a submodel whose output affects the dynamics of a node.
  - Examples: ionic currents, chemical reactions, hemodynamics, thermodynamics.
  - **Intrinsic mechanism** = a mechanism that depends only on the node it affects
  - **Connection mechanism** = a mechanism that depends on an additional source node

In DNSim, models are considered prescriptions for how the state of *things* (neutrally referred to as “nodes”) change over time. **Nodes** may be chemical species with states described by concentrations that are changed by chemical reactions, brain volumes with states described by BOLD signals that change with hemodynamics (or states described by temperatures that change according to thermodynamics), point neurons or neuron compartments with states described by voltages that change with ionic currents, or whatever else the modeler chooses to call a meaningful system/subsystem worth labeling. Note, while the term “node” may sound like we

are only discussing a single element, it may be better to think of “node” as “node type”; this is because each node has a number of copies (N), which can form a population of elements with equivalent models, when  $N > 1$ . Similarly, unless otherwise stated, when we talk below about source nodes, target nodes, etc, you should always assume each node has a variable N and, consequently, may represent a single element or a population of elements with equivalent models.

In DNSim, the way node states change over time is determined by the effects of “mechanisms” acting on the node. **Mechanisms** represent processes that change the state of a node (and may depend on the node state and their own internal state). Mechanisms can be considered modular submodels that may be associated with nodes arbitrarily and manipulated independently. In the above examples, chemical reactions, hemodynamics, thermodynamics, and ionic currents are all examples of potential mechanisms. Mechanisms only affect the node to which they’re added (i.e., associated) but may depend on the state of multiple nodes. If a mechanism only depends on the node it affects, it will be called an **intrinsic mechanism**; a mechanism that depends on multiple nodes pointing to the node it affects will be called a **connection mechanism**.

Mechanisms are the lowest-level component of the modeling hierarchy. Nodes are specified by listing the names of their intrinsic mechanisms (e.g.,  $E_{cell} = \{ Na, K, leak \}$ ). At the top of the hierarchy, **networks** are constructed by specifying connections from a source node to a target node by listing the names of their connection mechanisms (e.g.,  $E_{cell} \rightarrow I_{cell} = \{ AMPA, NMDA \}$ ). By using modular mechanisms and this hierarchical structure, given a database of existing mechanism models, the (potentially much) larger model can be fully specified easily by listing the mechanism names associated with nodes and the connections between nodes.